Viterbi Algorithm Generalized for n-Tape Best-Path Search

André Kempe

Xerox Research Centre Europe – Grenoble Laboratory
6 chemin de Maupertuis – 38240 Meylan – France

March 9, 2006

Abstract

We present a generalization of the Viterbi algorithm for identifying the path with minimal (resp. maximal) weight in a *n-tape weighted finite-state machine* (n-WFSM), that accepts a given n-tuple of input strings $\langle s_1, \dots s_n \rangle$. It also allows us to compile the best transduction of a given input n-tuple by a weighted $(n+m)$-WFSM (transducer) with n input and m output tapes. Our algorithm has a worst-case time complexity of $\mathcal{O}\,(\,|s|^n |E| \log |s|^n |Q|\,)$, where n and $|s|$ are the number and average length of the strings in the n-tuple, and $|Q|$ and $|E|$ the number of states and transitions in the n-WFSM, respectively. A straight forward alternative, consisting in intersection followed by classical shortest-distance search, operates in $\mathcal{O}\,(\,|s|^n (|E| + |Q|) \log |s|^n |Q|\,)$ time.

1 Introduction

The topic of this paper is situated in the areas of *multi-tape* or *n-tape weighted finite-state machines* (n-WFSMs) and shortest-path problems.

n-WFSMs (Rabin and Scott, 1959; Elgot and Mezei, 1965; Kay, 1987; Harju and Karhumäki, 1991; Kaplan and Kay, 1994) are a natural generalization of the familiar finite-state acceptors (one tape) and transducers (two tapes). The n-ary relation defined by an n-WFSM is a weighted *rational* relation. Finite relations are of particular interest since they can be viewed as relational databases. A finite-state transducer ($n = 2$) can be seen as a database of string pairs, such as $\langle$*spelling*, *pronunciation*$\rangle$ or $\langle$*French word*, *English word*$\rangle$. Unlike a classical database, a transducer may even define infinitely many pairs. For example, it may characterize the pattern of the spelling-pronunciation relationship in such a way that it can map even the spelling of an unknown word to zero or more possible pronunciations (with various weights), and vice-versa. n-WFSMs have been used in the morphological analysis of Semitic languages, to synchronize the vowels, consonants, and templatic pattern into a surface form (Kay, 1987; Kiraz, 2000).

Classical shortest-path algorithms can be separated into two groups, addressing either single-source shortest-path (SSSP) problems, such as Dijkstra's algorithm (Dijsktra, 1959) or Bellman-Ford's (Bellman, 1958; Ford and Fulkerson, 1956), or all-pairs shortest-path (APSP) problems, such as Floyd-Warshall's (Floyd, 1962; Warshall, 1962). SSSP algorithms determine a minimum-weight path from a source vertex of a real- or integer-weighted graph to all its other vertices. APSP algorithms find shortest paths between all pairs of vertices. For details of shortest-path problems in graphs see (Pettie, 2003), and in semiring-weighted finite-state automata see (Mohri, 2002).

We address the following problem: in a given n-WFSM we want to identify the path with minimal (resp. maximal) weight that accepts a given n-tuple of input strings $\langle s_1, \dots s_n \rangle$. This is of particular interest because it allows us also to compile the best transduction of a given input n-tuple by a weighted $(n+m)$-WFSM (transducer) with n input and m output tapes. For this, we identify the best path accepting the input n-tuple on its input tapes, and take the label of the path's output tapes as best output m-tuple.

A known straight forward method for solving our problem is to intersect the n-WFSM with another one that contains a single path labeled with the input n-tuple, and then to apply a classical SSSP algorithm, ignoring the labels. We show that such an intersection together with Dijkstra's algorithm have a worst-case time complexity of $\mathcal{O}\left(\,|s|^n(|E|+|Q|)\log|s|^n|Q|\,\right)$, where n and $|s|$ are the number and average length of the strings in the n-tuple, and $|Q|$ and $|E|$ the number of states and transitions of the n-WFSM, respectively.

We propose an alternative approach with lower complexity. It is based on the Viterbi algorithm which is generally used for detecting the most likely path in a *Hidden Markov Model* (HMM) for an observed sequence of symbols emitted by the HMM (Viterbi, 1967; Rabiner, 1990; Manning and Schütze, 1999). Our algorithm is a generalization of Viterbi's algorithm such that it deals with an n-tuple of input strings rather than with a single input string. In the worst case, it operates in $\mathcal{O}\left(\,|s|^n|E|\log|s|^n|Q|\,\right)$ time.

This paper is structured as follows. Basic definitions of weighted n-ary relations, n-WFSMs, HMMs, and the Viterbi algorithm are recalled in Section 2. Section 3 adapts the Viterbi algorithm to the search of the best path in a 1-WFSM that accepts a given input string, and Section 4 generalizes it to the search of the best path in an n-WFSM that accepts an n-tuple of strings. Section 5 illustrates our algorithm on a practical example, the alignment of word pairs (i.e., $n=2$), and provides test results that show a slightly higher than $\mathcal{O}\left(\,|s|^2\,\right)$ time complexity. The above mentioned classical method for solving our problem is discussed in Section 6. Section 7 concludes the paper.

2 Preliminaries

We recall some definitions about n-ary weighted relations and their machines, following the usual definitions for multi-tape automata (Elgot and Mezei, 1965; Eilenberg, 1974), with semiring weights added just as for acceptors and transducers (Kuich and Salomaa, 1986; Mohri, Pereira, and Riley, 1998). For more details see (Kempe, Champarnaud, and Eisner, 2004). We also briefly recall Hidden Markov Models and the Viterbi algorithm, and point the reader to (Viterbi, 1967; Rabiner, 1990; Manning and Schütze, 1999) for further details.

2.1 Weighted n-ary relations

A weighted n-ary relation is a function from $(\Sigma^*)^n$ to $\mathbb{K}$, for a given finite alphabet Σ and a given weight semiring $\mathcal{K}=\langle\mathbb{K},\oplus,\otimes,\bar{0},\bar{1}\rangle$. A relation assigns a weight to any n-tuple of strings. A weight of $\bar{0}$ can be interpreted as meaning that the tuple is not in the relation. We are especially interested in *rational* (or *regular*) n-ary relations, i.e. relations that can be encoded by n-tape weighted finite-state machines, that we now define.

We adopt the convention that variable names referring to n-tuples of strings include a superscript $^{(n)}$. Thus we write $s^{(n)}$ rather than $\vec{s}$ for a tuple of strings $\langle s_1,\ldots s_n\rangle$. We also use this convention for the names of objects that contain n-tuples of strings, such as n-tape machines and their transitions and paths.

2.2 Multi-tape weighted finite-state machines

An *n-tape weighted finite-state machine* (WFSM or n-WFSM) $A^{(n)}$ is defined by a six-tuple $A^{(n)}=\langle\Sigma,Q,\mathcal{K},E^{(n)},\lambda,\varrho\rangle$, with Σ being a finite alphabet, Q a finite set of states, $\mathcal{K}=\langle\mathbb{K},\oplus,\otimes,\bar{0},\bar{1}\rangle$ the semiring of weights, $E^{(n)}\subseteq(Q\times(\Sigma^*)^n\times\mathbb{K}\times Q)$ a finite set of weighted n-tape transitions, $\lambda:Q\to\mathbb{K}$ a function that assigns initial weights to states, and $\varrho:Q\to\mathbb{K}$ a function that assigns final weights to states.

Any transition $e^{(n)}\in E^{(n)}$ has the form $e^{(n)}=\langle y,\ell^{(n)},w,t\rangle$. We refer to these four components as the transition's source state $y(e^{(n)})\in Q$, its label $\ell(e^{(n)})\in(\Sigma^*)^n$, its weight $w(e^{(n)})\in\mathbb{K}$, and its

target state $t(e^{(n)}) \in Q$. We refer by $E(q)$ to the set of out-going transitions of a state $q \in Q$ (with $E(q) \subseteq E^{(n)}$).

A *path* $\gamma^{(n)}$ of length $k \geq 0$ is a sequence of transitions $e_1^{(n)} e_2^{(n)} \cdots e_k^{(n)}$ such that $t(e_i^{(n)}) = y(e_{i+1}^{(n)})$ for all $i \in [1, k-1]$. The label of a path is the element-wise concatenation of the labels of its transitions. The weight of a path $\gamma^{(n)}$ is

$$w(\gamma^{(n)}) \;=_{\text{def}}\; \lambda(y(e_1^{(n)})) \otimes \left(\bigotimes_{j \in [1,k]} w\left(e_j^{(n)}\right) \right) \otimes \varrho(t(e_k^{(n)})) \tag{1}$$

The path is said to be *successful*, and to *accept* its label, if $w(\gamma^{(n)}) \neq \bar{0}$.

2.3 Hidden Markov Models

A *Hidden Markov Model* (HMM) is defined by a five-tuple $\langle \Sigma, Q, \Pi, \mathsf{A}, \mathsf{B} \rangle$, where $\Sigma = \{\sigma_k\}$ is the output alphabet, $Q = \{q_i\}$ a finite set of states, $\Pi = \{\pi_i\}$ a vector of initial state probabilities $\pi_i = p(x_1 = q_i) : Q \to [0,1]$, $\mathsf{A} = \{\mathsf{a}_{ij}\}$ a matrix of state transition probabilities $\mathsf{a}_{ij} = p(x_t = q_j | x_{t-1} = q_i) : Q \times Q \to [0,1]$, and $\mathsf{B} = \{\mathsf{b}_{jk}\}$ a matrix of state emission probabilities $\mathsf{b}_{jk} = p(o_t = \sigma_k | x_t = q_j) : Q \times \Sigma \to [0,1]$. A *path* of length T in an HMM is a non-observable (i.e., hidden) state sequence $X = x_1 \cdots x_T$, emitting an observable output sequence $O = o_1 \cdots o_T$ which is a probabilistic function of X.

2.4 Viterbi Algorithm

The *Viterbi algorithm* finds the most likely path $\widehat{X} = \arg\max_X p(X|O, \mu)$ for an observed output sequence O and given model parameters $\mu = \langle \Pi, \mathsf{A}, \mathsf{B} \rangle$, using a trellis similar to that in Figure 1. It has a $\mathcal{O}(T\,|Q|^2)$ time and a $\mathcal{O}(T\,|Q|)$ space complexity.

3 1-Tape Best-Path Search

The Viterbi algorithm (Viterbi, 1967; Rabiner, 1990; Manning and Schütze, 1999) can be easily adapted for searching for the best of all paths of a 1-WFSM, $A^{(1)}$, that accept a given input string. We use a notation that will facilitate the subsequent generalization of the algorithm to n-tape best-path search (Section 4). Only the search for the path with minimal weight is explained. An adaptation to maximal weight search is trivial.

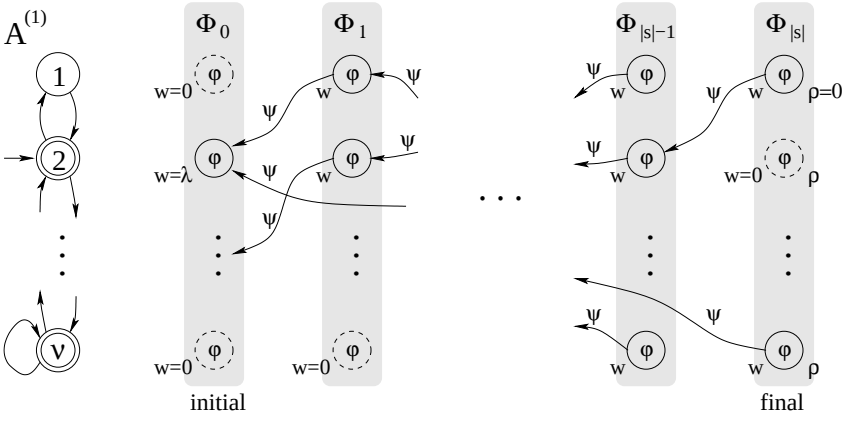

Figure 1: Modified trellis for 1-tape best-path search

3.1 Structures

We use a reading pointer $p \in P = \{0, \ldots |s|\}$ that is initially positioned before the first letter of the input string s, $p=0$, and then increased with the reading of s until it reaches the position after the last letter, $p=|s|$. At any moment, p equals the length of the prefix of s that has already been read.

As it is usual for the Viterbi algorithm, we use a trellis $\Phi = Q \times P$, consisting of nodes $\varphi = \langle q, p \rangle$ which express that a state $q \in Q$ is reached after reading p letters of s (Figure 1). We divide the trellis into several node sets $\Phi_p = \{\varphi = \langle q, p \rangle\} \subseteq \Phi$, each corresponding to a pointer position p or to a column of the trellis. For each node φ, we maintain three variables referring to φ's best prefix: w_φ being its weight, ψ_φ its last node (immediately preceding φ), and e_φ its last transition $e \in E$ of $A^{(1)}$. The ψ_φ are back-pointers that fully define the best prefix of each node φ. All w_φ, ψ_φ, and e_φ are initially undefined ($=\perp$).[1]

```
FsaViterbi(s, A^(1)) → γ :                                   [ γ = e_1 ··· e_r ]
1   Φ_initial ← ∅                                            [ Φ_initial = Φ_0 ]
2   for ∀q ∈ Q : λ(q) ≠ 0̄ do
3       φ ← ⟨q, 0⟩   ;   w_φ ← λ(q)   ;   Φ_initial ← Φ_initial ∪ {φ}
4   Φ ← {Φ_initial}
5   for p = 0, ... |s| − 1 do
6       for ∀φ = ⟨q, p⟩ ∈ Φ_p do
7           for ∀e ∈ E(q) do
8               if ∃u,v ∈ Σ* : uℓ(e)v = s ∧ p = |u|
9                   then p' ← p + |ℓ(e)|
10                      φ' ← ⟨t(e), p'⟩   ;   w' ← w_φ ⊗ w(e)
11                      Φ ← Φ ∪ {Φ_p'}
12                      Φ_p' ← Φ_p' ∪ {φ'}
13                      if w_φ' = ⊥ ∨ w_φ' > w'
14                          then w_φ' ← w'   ;   ψ_φ' ← φ   ;   e_φ' ← e
15  φ̂ ← arg min_{φ=⟨q,p⟩ ∈ Φ_final} (w_φ ϱ(q))                [ Φ_final = Φ_|s| ]
16  γ ← getPath(φ̂)
17  return γ
```

Figure 2: Pseudocode of 1-tape best-path search

3.2 Algorithm

The algorithm FsaViterbi( ) returns from all paths γ of the 1-WFSM $A^{(1)}$ that accept the string s, the one with minimal weight (Figure 2). $A^{(1)}$ must not contain any transitions labeled with ε (the empty string). At least a partial order must be defined on the semiring of weights. Nothing else is required concerning the labels, weights, or structure of $A^{(1)}$.[2]

The algorithm starts with creating an initial node set $\Phi_{\text{initial}} = \Phi_0$ for the initial position $p = 0$ of the reading pointer. The set Φ_{initial} contains a node for each initial state of $A^{(1)}$ (Lines 1–3). The prefix weights w_φ of these nodes are set to the initial weight $\lambda(q)$ of the respective states q. The set of node sets $\mathbf{\Phi}$ contains only Φ_{initial} at this point (Line 4).

[1] The variables w_φ, ψ_φ, and e_φ can be formally regarded as elements of the vectors $\overline{w}$, $\overline{\psi}$, and $\overline{e}$, respectively, that are indexed by values of φ. In a practical implementation is, however, meaningful to store these variables directly on the node that they refer to.

[2] Cycles are, e.g., not required to have non-negative weights (as for Dijkstra's algorithm) because all paths of interest are constrained by the input string.

In the subsequent iteration (Lines 5–14), reaching from the first to the one but last pointer position, $p = 0, \ldots |s|-1$, we inspect all outgoing transitions $e \in E(q)$ of all states $q \in Q$ for which there is a node $\varphi = \langle q, p \rangle$ in Φ_p. If the label $\ell(e)$ of e matches s at position p, we create a new node $\varphi' = \langle t(e), p' \rangle$ for the target $t(e)$ of e (Line 6). Its prefix weight w' equals the current node's weight w_φ multiplied by the weight $w(e)$ of e. The node set $\Phi_{p'}$ for the new φ' is created and inserted into the set of node sets $\mathbf{\Phi}$ (if it does not exist yet; Line 11). Then φ' is inserted into $\Phi_{p'}$ (if it is not yet a member of it; Line 12). If the prefix weight of φ' is still undefined, $w_{\varphi'} = \perp$ (because no prefix of φ' has been analyzed yet), or if it is higher than the weight of the currently analyzed new prefix, $w_{\varphi'} > w'$, then the variables $w_{\varphi'}$, $\psi_{\varphi'}$, and $e_{\varphi'}$ of φ' are assigned values of the new prefix (Lines 13–14).

The algorithm terminates by selecting the node $\widehat{\varphi}$, corresponding to the path with the minimal weight, from the final node set $\Phi_{\mathsf{final}} = \Phi_{|s|}$. This weight is the product of the node's prefix weight w_φ and the final weight $\varrho(q)$ of the corresponding state $q \in Q$ (Line 15). The function *getPath*( ) identifies the best path γ by following all back-pointers ψ_φ, from the node $\widehat{\varphi} \in \Phi_{\mathsf{final}}$ to some node $\varphi \in \Phi_{\mathsf{initial}}$, and collecting all transitions $e = e_\varphi$ it encounters. Finally, γ is returned.

3.3 ε-Transitions

The algorithm can be extended to allow for ε-transitions (but not for ε-cycles). The source and target node, φ and φ', of an ε-transition would be in the same Φ_p. If $\varphi' = \langle q', p' \rangle$ is actually inserted into Φ_p (Line 12) or if its variables $w_{\varphi'}$, $\psi_{\varphi'}$, and $e_{\varphi'}$ change their values (Lines 13–14), then we have to (re-)"include" φ' into the iteration over all nodes of the currently inspected Φ_p (Line 6). The algorithm will still terminate since there can be only finite sequences of ε-transitions (as long as we have no ε-cycles).

3.4 Best transduction

The algorithm FsaViterbi( ) can be used for compiling the best transduction of a given input string s by a 2-WFSM (weighted transducer). For this, we identify the best path γ accepting s on its input tape and take the label of γ's output tape as best output string v.

4 n-Tape Best-Path Search

We come now to the central topic of this paper: the generalization of the Viterbi algorithm for searching for the best of all paths of an n-WFSM, $A^{(n)}$, that accept a given n-tuple of input strings, $s^{(n)} = \langle s_1, \ldots s_n \rangle$. This requires relatively few modifications to the above explained structures and algorithm (Section 3).

4.1 Structures

The main difference wrt. the previous structures is that now our reading pointer is a vector of n natural integers, $p^{(n)} = \langle p_1, \ldots p_n \rangle \in \big([0, \ldots |s_1|] \times \ldots \times [0, \ldots |s_n|]\big) \subset \mathbb{N}^n$. The pointer is initially positioned before the first letter of each s_i $(\forall i \in [1, n])$, $p^{(n)} = \langle 0, \ldots 0 \rangle$. Its elements p_i are then increased according to the non-synchronized reading of the s_i on the tapes i $(\forall i \in [1, n])$, until the pointer reaches its final position after the last letter of each s_i, $p^{(n)} = \langle |s_1|, \ldots |s_n| \rangle$.

More precisely, a pointer is an element of the monoid $\langle \mathbb{N}^n, +, \mathbf{0} \rangle$ with $+$ being vector addition and $\mathbf{0}$ the vector of n 0's. We have a partial order of pointers. Let $\sqsubset : \mathbb{N}^n \times \mathbb{N}^n \to \{\text{TRUE}, \text{FALSE}\}$. Let $a, b \in \mathbb{N}^n$, then $a \sqsubset b \iff (\exists c \in \mathbb{N}^n, c \neq \mathbf{0} : a + c = b)$. We say *a precedes b*. It holds that $a \sqsubset b \Rightarrow (\sum_{i=1}^n a_i < \sum_{i=1}^n b_i)$ where a_i and b_i are the vector elements.

In the trellis (Figure 3) we have still one node set $\Phi_{p^{(n)}}$ per pointer position $p^{(n)}$, a single initial node set $\Phi_{\mathsf{initial}} = \Phi_{\langle 0, \ldots 0 \rangle}$ and a single final node set $\Phi_{\mathsf{final}} = \Phi_{\langle |s_1|, \ldots |s_n| \rangle}$. There are, however, several

nodes sets in parallel between the two (corresponding to pointers $p^{(n)}, p'^{(n)}$ not preceding each other, i.e., $p^{(n)} \not\sqsubset p'^{(n)} \wedge p'^{(n)} \not\sqsubset p^{(n)}$).

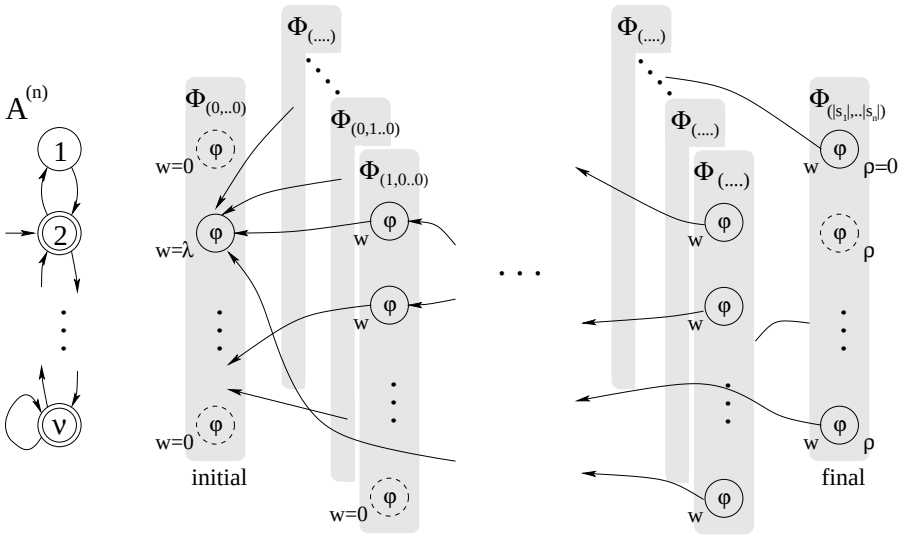

Figure 3: Modified trellis for n-tape best-path search

4.2 Algorithm

The algorithm FsmViterbi( ) returns from all paths $\gamma^{(n)}$ of the n-WFSM $A^{(n)}$ that accept the string tuple $s^{(n)}$, the one with minimal weight (Figure 4). $A^{(n)}$ must not contain any transitions labeled with $\langle \varepsilon, \ldots \varepsilon \rangle$.[3]

The initial node set $\Phi_{\mathsf{initial}} = \Phi_{\langle 0,\ldots 0 \rangle}$ is created as before, and inserted into the set of node sets $\mathbf{\Phi}$ (Lines 1–4). In addition, it is inserted into a Fibonacci heap[4] $\mathbf{H}$ (Line 4) (Fredman and Tarjan, 1987). This heap contains node sets $\Phi_{p^{(n)}}$ that have not yet been processed, and uses $\sum_{i=1}^{n} p_i$ as sorting key.

The subsequent iteration continues as long as $\mathbf{H}$ is not empty (Lines 5–16). The function *extract-MinElement*( ) extracts the (or a) minimal element $\Phi_{p^{(n)}}$ from $\mathbf{H}$ (Line 6). Due to our sorting key, none of the remaining $\Phi_{p'^{(n)}}$ in $\mathbf{H}$ is a predecessor to $\Phi_{p^{(n)}}$: $\forall \Phi_{p'^{(n)}} \in \mathbf{H}$, $p'^{(n)} \not\sqsubset p^{(n)}$. This property prevents the compilation of suffixes of a $\Phi_{p^{(n)}}$ that has some not yet analyzed prefixes (which could lead to wrong choices). The extracted $\Phi_{p^{(n)}}$ is handled almost as in the previous algorithm (Figure 2). Transition labels $\ell(e^{(n)})$ are required to match with a factor of $s^{(n)}$ at position $p^{(n)}$ (Line 9). New $\Phi_{p'^{(n)}}$ are inserted both into $\mathbf{\Phi}$ and $\mathbf{H}$ (Lines 12–13).

4.3 Best transduction

The algorithm FsmViterbi( ) can be used for obtaining from a weighted (n+m)-WFSM (transducer) with n input and m output tapes, the best transduction of a given input n-tuple $s^{(n)}$. For this, we identify the best path $\gamma^{(n+m)}$ accepting $s^{(n)}$ on its n input tapes and take the label of γ's m output tapes as best output m-tuple $v^{(m)}$. Input and output tapes can be in any order.

[3] The algorithm can be extended to allow for $\langle \varepsilon, \ldots \varepsilon \rangle$-transitions (but not for $\langle \varepsilon, \ldots \varepsilon \rangle$-cycles) as described in Section 3.

[4] Alternatively, one could use a binary heap. Tests on a concrete example have, however, shown that the algorithm performs slightly better with a Fibonacci heap (Table 1).

```
FsmViterbi(s^(n), A^(n)) → γ^(n) :                                  [ γ^(n) = e_1^(n) ··· e_r^(n) ]
1   Φ_initial ← ∅                                                   [ Φ_initial = Φ_⟨0,...0⟩ ]
2   for ∀q ∈ Q : λ(q) ≠ 0̄ do
3       φ ← ⟨q, ⟨0, ... 0⟩⟩   ;   w_φ ← λ(q)   ;   Φ_initial ← Φ_initial ∪ {φ}
4   Φ ← {Φ_initial}   ;   H ← {Φ_initial}
5   while H ≠ ∅ do
6       Φ_p(n) ← extractMinElement(H)
7       for ∀φ = ⟨q, p^(n)⟩ ∈ Φ_p(n) do
8           for ∀e^(n) ∈ E(q) do
9               if ∃u^(n),v^(n) ∈ (Σ*)^n : u^(n) ℓ(e^(n)) v^(n) = s^(n) ∧ p^(n) = ⟨|u_1|, ... |u_n|⟩
10                  then p'^(n) ← p^(n) + ⟨|(ℓ(e^(n)))_1|, ... |(ℓ(e^(n)))_n|⟩
11                      φ' ← ⟨t(e^(n)), p'^(n)⟩   ;   w' ← w_φ ⊗ w(e^(n))
12                      if Φ_p'(n) ∉ Φ
13                          then Φ ← Φ ∪ {Φ_p'(n)}   ;   H ← H ∪ {Φ_p'(n)}
14                      Φ_p'(n) ← Φ_p'(n) ∪ {φ'}
15                      if w_φ' = ⊥ ∨ w_φ' > w'
16                          then w_φ' ← w'   ;   ψ_φ' ← φ   ;   e_φ' ← e^(n)
17  φ̂ ← arg min_{φ=⟨q,p^(n)⟩ ∈ Φ_final} (w_φ ϱ(q))                  [ Φ_final = Φ_⟨|s_1|,...|s_n|⟩ ]
18  γ^(n) ← getPath(φ̂)
19  return γ^(n)
```

Figure 4: Pseudocode of n-tape best-path search

4.4 Complexity

The trellis (Figure 3) consists of at most $|P| = \prod_{i=1}^{n}(|s_i|+1)$ node sets $\Phi_{p^{(n)}} \in \mathbf{\Phi}$. Assuming approximately equal length $|s|$ for all s_i of $s^{(n)}$, we can simplify: $|P| \approx (|s|+1)^n$. For each node set $\Phi_{p^{(n)}}$ we have to create at most $|Q|$ nodes $\varphi \in \Phi_{p^{(n)}}$, which leads to a $\mathcal{O}(|s|^n|Q|)$ space complexity for our algorithm.

Each $\Phi_{p^{(n)}}$ is extracted once from the Fibonacci heap $\mathbf{H}$ in $\mathcal{O}(\log|P|)$ time. We analyze for $\Phi_{p^{(n)}}$ at most $|E|$ transitions $e \in E$ of $A^{(n)}$. For the target of each e we find a $\Phi_{p'^{(n)}} \in \mathbf{\Phi}$ in $\mathcal{O}(\log|P|)$ time and a node $\varphi' \in \Phi_{p'^{(n)}}$ in $\mathcal{O}(\log|Q|)$ time. Thus, FsmViterbi( ) has a worst-case overall time complexity of $\mathcal{O}\left(\,|P|(\log|P| + |E|(\log|P| + \log|Q|))\,\right) = \mathcal{O}\left(\,|P||E|\log|P||Q|\,\right) = \mathcal{O}\left(\,|s|^n|E|\log|s|^n|Q|\,\right)$.

An HMM has exactly one transition per state pair, so that $|E| = |Q|^2$, and an arity of $n = 1$. There would also be never more than one $\Phi_{p^{(n)}}$ on the heap, extractable in constant time. In this case, our algorithm has a $\mathcal{O}(|s||Q|)$ space and a $\mathcal{O}\left(|s||Q|^2\right)$ time complexity, as has the classical version of the Viterbi algorithm (Section 2).

5 Example: Word Alignment

In this section we illustrate our n-tape best path search on a practical example: the alignment of word pairs.

Suppose, we want to create a (non-weighted) transducer, $D^{(2)}$, from a list of word pairs $s^{(2)}$ of the form $\langle \textit{inflected form}, \textit{lemma} \rangle$, e.g., $\langle \mathsf{swum}, \mathsf{swim} \rangle$, such that each path of the transducer is labeled with one of the pairs. We want to use only transition labels of the form $\langle \sigma, \sigma \rangle$, $\langle \sigma, \varepsilon \rangle$, or $\langle \varepsilon, \sigma \rangle$ $(\forall \sigma \in \Sigma)$, while keeping paths as short as possible. For example, $\langle \mathsf{swum}, \mathsf{swim} \rangle$ should be encoded either by the sequence $\langle \mathsf{s}, \mathsf{s} \rangle \langle \mathsf{w}, \mathsf{w} \rangle \langle \mathsf{u}, \varepsilon \rangle \langle \varepsilon, \mathsf{i} \rangle \langle \mathsf{m}, \mathsf{m} \rangle$ or by $\langle \mathsf{s}, \mathsf{s} \rangle \langle \mathsf{w}, \mathsf{w} \rangle \langle \varepsilon, \mathsf{i} \rangle \langle \mathsf{u}, \varepsilon \rangle \langle \mathsf{m}, \mathsf{m} \rangle$, rather than by the ill-formed

$\langle \mathsf{s},\mathsf{s}\rangle\langle \mathsf{w},\mathsf{w}\rangle\langle \mathsf{u},\mathsf{i}\rangle\langle \mathsf{m},\mathsf{m}\rangle$, or the sub-optimal $\langle \mathsf{s},\varepsilon\rangle\langle \mathsf{w},\varepsilon\rangle\langle \mathsf{u},\varepsilon\rangle\langle \mathsf{m},\varepsilon\rangle\langle \varepsilon,\mathsf{s}\rangle\langle \varepsilon,\mathsf{w}\rangle\langle \varepsilon,\mathsf{i}\rangle\langle \varepsilon,\mathsf{m}\rangle$. To achieve this, we perform for each word pair an alignment based on minimal edit distance.

5.1 Standard solution with edit distance matrix

A well known standard solution for word alignment is based on edit distance which is a string similarity measure defined as the minimum cost needed to convert one string into another (Wagner and Fischer, 1974; Pirkola et al., 2003).

For two words, $a = a_1 \dots a_n$ and $b = b_1 \dots b_m$, the edit distance can be compiled with a matrix $X = \{x_{i,j}\}$ $(i \in [0,n],\ j \in [0,m])$ (Figures 5 and 6). A horizontal move in X at a cost c_I expresses an *insertion*, a vertical move at a cost c_D a *deletion*, and a diagonal move at a cost c_S a *substitution* if $a_i \neq b_j$ or no edit operation if $a_i = b_j$. We set $c_I = c_D = 1$, $c_S = \infty$ for $a_i \neq b_j$ (to disable substitutions), and $c_S = 0$ for $a_i = b_j$. The element $x_{0,0}$ is set to 0 and all other $x_{i,j}$ to $\min(x_{i,j-1} + c_I\ ,\ x_{i-1,j} + c_D\ ,\ x_{i-1,j-1} + c_S)$, insofar as these choices are available, proceeding top-down and left-to-right. The choices made to go from $x_{0,0}$ to $x_{n,m}$ describe the set of paths with (the same) minimal cost. Each of these paths defines a sequence of edit operations for transforming a into b.

The algorithm operates in $\mathcal{O}(|a||b|)$ time and space complexity.

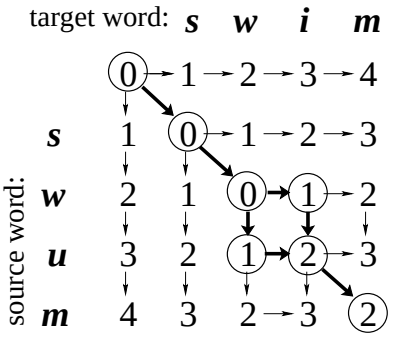

Figure 5: Edit distance matrix $X = \{x_{i,j}\}$ (choices are indicated by arrows; minimum cost paths by thick arrows and circles)

```
1   x_{0,0} ← 0
2   for i = 1 ... |a|  do
3       x_{i,0} ← x_{i-1,0} + c_D
4   for j = 1 ... |b|  do
5       x_{0,j} ← x_{0,j-1} + c_I
6   for i = 1 ... |a|  do
7       for j = 1 ... |b|  do
8           m_D ← x_{i-1,j} + c_D
9           m_I ← x_{i,j-1} + c_I
10          m_S ← x_{i-1,j-1} + c_S
11          x_{i,j} ← min( m_D, m_I, m_S )
```

Figure 6: Pseudocode of compiling an *edit distance matrix*

5.2 Solution with 2-tape best path search

Alternatively, word alignment can be performed by best path search on an n-WFSM, such as $A^{(5)}$ generated from the expression (Isabelle and Kempe, 2004)

$$A^{(5)} \quad = \quad \Big(\langle\langle \mathtt{?},\mathtt{?},\mathtt{?},\mathtt{?},\mathtt{K}\rangle_{\{1=2=3=4\}}, 0\rangle \\ \cup\ \langle\langle \varepsilon,\mathtt{?},@,\mathtt{?},\mathtt{I}\rangle_{\{2=4\}}, 1\rangle\ \cup\ \langle\langle \mathtt{?},\varepsilon,\mathtt{?},@,\mathtt{D}\rangle_{\{1=3\}}, 1\rangle\ \Big)^* \qquad (2)$$

where ? can be instantiated by any symbol $\sigma \in \Sigma$, @ is a special symbol representing ε in an alignment, $\{1=2=3=4\}$ a constraint requiring the ?'s on tapes 1 to 4 to be instantiated by the same symbol (Nicart et al., 2006),[5] and 0 and 1 are weights over the semiring $\langle \mathbb{N} \cup \{\infty\}, \min, +, \infty, 0\rangle$.

Input word pairs $s^{(2)} = \langle s_1, s_2\rangle$ will be matched on tape 1 and 2, and aligned output word pairs generated from tape 3 and 4. A symbol pair $\langle \mathtt{?},\mathtt{?}\rangle$ read on tape 1 and 2 is identically mapped to $\langle \mathtt{?},\mathtt{?}\rangle$ on tape 3 and 4, a $\langle \varepsilon,\mathtt{?}\rangle$ is mapped to $\langle @,\mathtt{?}\rangle$, and a $\langle \mathtt{?},\varepsilon\rangle$ to $\langle \mathtt{?},@\rangle$. $A^{(5)}$ will introduce @'s in s_1 (resp.

[5] Roughly following (Kempe, Champarnaud, and Eisner, 2004), we employ here a simpler notation for constraints than in (Nicart et al., 2006).

in s_2) at positions where $D^{(2)}$ shall have $\langle\varepsilon,\sigma\rangle$- (resp. a $\langle\sigma,\varepsilon\rangle$-) transitions. (Later, we simply replace in $D^{(2)}$ all @ by ε.)

Thus, we obtain the full set of all possible alignments between s_1 and s_2. The best alignment is the one with the lowest weight. For example, $\langle$swum, swim$\rangle$ is mapped to a set of alignments, including the two best ones, $\langle$sw@um, swi@m$\rangle$ and $\langle$swu@m, sw@im$\rangle$, with weight 2 both. The (or a) best alignment can be found without generating all alignments, by means of our n-tape best path search (with $n=2$).

So far, we did not use tape 5. It can serve for excluding certain paths. For example, joining $A^{(5)}$ on tape 5 with $C^{(1)}$ (Kempe et al., 2005a; Kempe et al., 2005b) built from the expression $\neg($`?* I D ?*`$)$, prohibiting an insertion (`I`) to be immediately followed by a deletion (`D`), would leave only $\langle$swu@m, sw@im$\rangle$ as a best path.

The 5-WFSM from Equation (2) has 1 state and 3 transitions. Input is read on 2 tapes. Our algorithm works on this example with a worst-case time complexity of $\mathcal{O}(\,|s_1||s_2|\cdot 3\cdot\log(|s_1||s_2|\cdot 1)\,) = \mathcal{O}(\,|s_1||s_2|\log|s_1||s_2|\,)$ and a worst-case space complexity of $\mathcal{O}(\,|s_1||s_2|\cdot 1\,) = \mathcal{O}(\,|s_1||s_2|\,)$.

5.3 Test results

We tested our n-tape best-path algorithm on the alignment of the German word pair $\langle$gemacht, machen$\rangle$ (English: $\langle$*done*, *do*$\rangle$), leading to $\langle$gemacht@@, @@mach@en$\rangle$. We repeated this test for the word pairs $\langle s_1^r, s_2^r\rangle$ with $s_1=$"gemacht" and $s_2=$"machen", and $r\in[1,8]$.[6]

r	A	B	C	D
1	1	1	1	1.056
2	4	4.12	5.48	1.041
3	9	9.41	14.3	1.057
4	16	17.1	27.9	1.029
5	25	27.2	46.5	1.059
6	36	39.8	70.5	1.016
7	49	54.1	100	1.005
8	64	70.8	135	1.006

Table 1: Test results for word pair alignment with 2-tape best path search

The columns of Table 1 show for different r :

(A) an estimated time ratio of r^2 for the classical approach with an edit distance matrix,

(B) the measured time ratio for 2-tape best path search (wrt. 3.93 milliseconds for $r=1$) using a Fibonacci heap,

(C) an estimated worst-case time ratio of $\frac{(7r\cdot 6r)\log(7r\cdot 6r)}{(7\cdot 6)\log(7\cdot 6)} = r^2(1+2\frac{\log r}{\log 42})$ corresponding to the worst-case complexity of $\mathcal{O}(7r6r\log 7r6r)$ for the two words of length $7r$ and $6r$, respectively, and

(D) the measured time increase factor when using a binary instead of a Fibonacci heap.

Comparing the columns A and B shows a time complexity slightly above $\mathcal{O}(r^2) = \mathcal{O}(\,|s_1^r||s_2^r|\,)$, being much lower than the worst-case time complexity in column C, for our algorithm on this example.

[6]For example, for $r=2$ we have $\langle$gemachtgemacht, machenmachen$\rangle$.

6 An Alternative Approach

A well-known straight forward alternative to the above n-tape best-path search on an n-WFSM $A^{(n)}$ is to intersect $A^{(n)}$ with an n-WFSM $I^{(n)}$, containing a single path labeled with the input n-tuple $s^{(n)}$, and then to apply a classical shortest-distance algorithm, ignoring the labels.

6.1 Intersection

The intersection $B^{(n)} = I^{(n)} \cap A^{(n)}$ can be compiled as the join $I^{(n)} \bowtie_{\{1=1,\dots n=n\}} A^{(n)}$ (Kempe, Champarnaud, and Eisner, 2004). In general, it has undecidable emptiness and rationality (Rabin and Scott, 1959). In our case, however, with $A^{(n)}$ being $\langle\varepsilon,\dots\varepsilon\rangle$-cycle free and $I^{(n)}$ acyclic, it is even for non-commutative semirings always rational.[7]

Actually, the trellis Φ in Figure 3 corresponds partially to $B^{(n)}$. Each node $\varphi \in \Phi$ corresponds to a state $q \in Q_B$ of $B^{(n)}$ (and vice versa); however, only those transitions $e \in E_B$ of $B^{(n)}$ that correspond to a state's best prefix, occur as "best transitions" e_φ in Φ.[8]

From this analogy we deduce that compiling the intersection $B^{(n)}$ has a worst-case time and space complexity of $\mathcal{O}\left(\,|P||E|\log|P||Q|\,\right)$, with $|P| = (|s|+1)^n$, equal to the time complexity for constructing the trellis. The result, $B^{(n)}$, has at most $\nu \leq |P||Q|$ states and $\mu \leq |P||E|$ transitions.

6.2 Shortest-distance algorithms

Since any n-WFSM with multiple initial states can be transformed into one with a single initial state, we can use any algorithm that solves a single-source shortest-distance problem, such as Dijkstra's algorithm (Dijsktra, 1959) combined with Fibonacci heaps (Fredman and Tarjan, 1987), that operates in $\mathcal{O}(\mu + \nu\log\nu)$ time, or Bellman-Ford's algorithm (Bellman, 1958; Ford and Fulkerson, 1956) operating in $\mathcal{O}(\mu\nu)$ time, with ν being the number of states and μ the number of transitions.

Recently, it has been shown that any single-source shortest-distance algorithm on directed graphs has a lower bound of $\Omega(\mu + \min(\nu\log\nu,\ \nu\log\rho))$ where ρ is the ratio of the maximal to minimal transition weight (Pettie, 2003). Since we cannot make any assumption concerning ρ in general, we consider $\widehat{\Omega}(\mu + \nu\log\nu)$ as a "worst-case lower bound". It equals the upper bound of Dijkstra's algorithm.

On the intersection $B^{(n)} = I^{(n)} \cap A^{(n)}$, Dijkstra's algorithm requires $\mathcal{O}(|P||E| + |P||Q|\log|P||Q|)$ time, and Bellman-Ford's $\mathcal{O}(|P|^2|E||Q|)$ time, in the worst case. The sets E and Q refer to $A^{(n)}$.

6.3 Complete estimate

Intersection and Dijkstra's algorithm have together a worst-case time complexity of $\mathcal{O}\left(\,|P||E|\log|P||Q| + |P||E| + |P||Q|\log|P||Q|\,\right) \approx \mathcal{O}\left(\,|P|(|E|+|Q|)\log|P||Q|\,\right)$. For intersection and Bellman-Ford's algorithm it is $\mathcal{O}\left(\,|P||E|\log|P||Q| + |P|^2|E||Q|\,\right) = \mathcal{O}\left(\,|P||E|\,(|P||Q| + \log|P||Q|)\,\right)$. Both combinations exceed the complexity of our algorithm.

This result is not surprising since only building the trellis Φ should take less time than building the intersection $B^{(n)}$ (which is a kind of "superset" of Φ) and then performing a best-path search.

[7] The intersection of two n-WFSM over non-commutative semirings is in general not rational (even for $n=1$).

[8] Due to this analogy, one can easily derive an n-tape intersection (or join) algorithm, for precisely our case, from the algorithm in Figure 4. Trellis nodes would become states of the resulting n-WFSM. All of their incoming transitions would be constructed, rather than only those that correspond to a best prefix. The state set would be partitioned like the trellis. The Fibonacci heap can be replaced by a stack (which does not decrease the overall time complexity), because the order in which partitions are treated would be irrelevant.

7 Conclusion

We presented an algorithm for identifying the path with minimal (resp. maximal) weight in a given *n-tape weighted finite-state machine* (n-WFSM), $A^{(n)}$, that accepts a given n-tuple of input strings, $s^{(n)} = \langle s_1, \ldots s_n \rangle$. This problem is of particular interest because it allows us also to compile the best transduction of a given input n-tuple $s^{(n)}$ by a weighted $(n+m)$-WFSM (transducer), $A^{(n+m)}$, with n input and m output tapes. For this, we identify the best path accepting $s^{(n)}$ on its n input tapes, and take the label of its output tapes as best output m-tuple $v^{(m)}$. (Input and output tapes can be in any order.)

Our algorithm is a generalization of the Viterbi algorithm which is generally used for detecting the most likely path in a *Hidden Markov Model* (HMM) for an observed sequence of symbols emitted by the HMM. In the worst case, it operates in $\mathcal{O}\left(|s|^n |E| \log |s|^n |Q| \right)$ time, where n and $|s|$ are the number and average length of the strings in $s^{(n)}$, and $|Q|$ and $|E|$ the number of states and transitions of $A^{(n)}$, respectively.

We illustrated our n-tape best path search on a practical example, the alignment of word pairs (i.e., $n=2$), and provided test results that show a time complexity slightly higher than $\mathcal{O}\left(|s|^2 \right)$.

Finally, we discussed a straight forward alternative approach for solving our problem, that consists in intersecting $A^{(n)}$ with an n-WFSM $I^{(n)}$, that has a single path labeled with the input n-tuple $s^{(n)}$, and then applying a classical shortest-distance algorithm, ignoring the labels. This has, however, a worst-case time complexity of $\mathcal{O}\left(|s|^n (|E| + |Q|) \log |s|^n |Q| \right)$, which is higher than that of our algorithm.

References

Bellman, Richard. 1958. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90.

Dijsktra, Edsger W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.

Eilenberg, Samuel. 1974. *Automata, Languages, and Machines*, volume A. Academic Press, San Diego.

Elgot, Calvin C. and Jorge E. Mezei. 1965. On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1):47–68.

Floyd, Robert W. 1962. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345.

Ford, Lester R. and Delbert R. Fulkerson. 1956. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):99–404.

Fredman, Michael L. and Robert Endre Tarjan. 1987. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615.

Harju, Tero and Juhani Karhumäki. 1991. The equivalence problem of multitape finite automata. *Theoretical Computer Science*, 78(2):347–355.

Isabelle, Pierre and André Kempe. 2004. Automatic string alignment for finite-state transducers. *Unpublished work*.

Kaplan, Ronald M. and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

Kay, Martin. 1987. Nonconcatenative finite-state morphology. In *Proc. 3rd Int. Conf. EACL*, pages 2–10, Copenhagen, Denmark.

Kempe, André, Jean-Marc Champarnaud, and Jason Eisner. 2004. A note on join and auto-intersection of n-ary rational relations. In B. Watson and L. Cleophas, editors, *Proc. Eindhoven FASTAR Days*, number 04–40 in TU/e CS TR, pages 64–78, Eindhoven, Netherlands.

Kempe, André, Jean-Marc Champarnaud, Jason Eisner, Franck Guingne, and Florent Nicart. 2005a. A class of rational n-wfsm auto-intersections. In J. Farré, I. Litovski, and S. Schmitz, editors, *Proc. 10th Int. Conf. on Implementation and Application of Automata (CIAA'05)*, pages 266–274, Sophia Antipolis, France.

Kempe, André, Jean-Marc Champarnaud, Franck Guingne, and Florent Nicart. 2005b. Wfsm auto-intersection and join algorithms. In *Proc. 5th Int. Workshop on Finite-State Methods and Natural Language Processing (FSMNLP'05)*, Helsinki, Finland.

Kiraz, George Anton. 2000. Multitiered nonlinear morphology using multitape finite automata: a case study on Syriac and Arabic. *Computational Lingistics*, 26(1):77–105, March.

Kuich, Werner and Arto Salomaa. 1986. *Semirings, Automata, Languages*. Number 5 in EATCS Monographs on Theoretical Computer Science. Springer Verlag, Berlin, Germany.

Manning, Christopher D. and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA.

Mohri, Mehryar. 2002. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350.

Mohri, Mehryar, Fernando C. N. Pereira, and Michael Riley. 1998. A rational design for a weighted finite-state transducer library. *Lecture Notes in Computer Science*, 1436:144–158.

Nicart, Florent, Jean-Marc Champarnaud, Tibor Csáki, Tamás Gaál, and André Kempe. 2006. Multi-tape automata with symbol classes. In O.H. Ibarra and H.-C. Yen, editors, *Proc. 11th Int. Conf. on Implementation and Application of Automata (CIAA'06)*, volume 4094 of *Lecture Notes in Computer Science*, pages 126–136, Taipei, Taiwan. Springer Verlag.

Pettie, Seth. 2003. A new approach to all-pairs shortest paths on real-weighted graphs. *Theoretical Computer Science*, 312(1):47–74. special issue of selected papers from ICALP 2002.

Pirkola, Ari, Jarmo Toivonen, Heikki Keskustalo, Kari Visala, and Kalervo Järvelin. 2003. Fuzzy translation of cross-lingual spelling variants. In *Proceedings of the 26th Annual International ACM SIGIR*, pages 345–352, Toronto, Canada.

Rabin, Michael O. and Dana Scott. 1959. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125.

Rabiner, Lawrence R. 1990. A tutorial on hidden markov models and selected applications in speech recognition. In Alex Waibel and Kai-Fu Lee, editors, *Readings in Speech Recognition*. Morgan Kaufmann, pages 267–296.

Viterbi, Andrew J. 1967. Error bounds for convolutional codes and an asymptotical optimal decoding algorithm. In *Proceedings of the IEEE*, volume 61, pages 268–278. Institute of Electrical and Electronics Engineers.

Wagner, Robert A. and Michael J. Fischer. 1974. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168–173.

Warshall, Stephan. 1962. A theorem on boolean matrices. *Journal of the ACM*, 9(1):11–12.